Chapitre 2 : **Les fichiers**

1. Introduction
2. Définition
3. Types de fichier
4. Manipulation des fichiers

Définition

Des fois, il est nécessaire de conserver certaines données après la fin du programme, ceci pour une utilisation future, les fichiers ont été conçus pour ces fins.

Un **fichier** est une structure de données, toutes de même type. L'accès à un élément (une donnée) du fichier peut se faire :

- De manière séquentielle : c'est-à-dire en parcourant le fichier élément par élément depuis le début jusqu'à l'élément choisi ;
- De manière direct : en donnant la position de l'élément ;

Les fichiers sont conservés en **mémoire secondaire** (disques, flash disk, ...), les données qui les constituent reste toujours tant que cette mémoire secondaire n'est pas formatée ou endommagée. Chaque fichier est désigné par un nom et possède des attributs tels que date de création, taille, icône...

Il existe deux types de fichiers :

1. **Les fichiers binaires** : Contenant du code binaire représentant chaque élément. Ces fichiers ne peuvent être manipulés que par des programmes!
2. **Les fichiers texte** : appelés aussi imprimables, contenant des caractères et susceptibles d'être lus, éditées, imprimés....

Operations sur les fichiers

Création :

Pour des raisons d'efficacité, les accès à un fichier se font par l'intermédiaire d'une mémoire-tampon (buffer), ce qui permet de réduire le nombre d'accès aux périphériques (disque...).

Déclaration d’une variable fichier (binaire ou texte)

*FILE *fichier ;*

« fichier » est le nom de la variable.

Un objet de type FILE * est appelé flot de données (en anglais, stream).

Remarque :

Attention aux majuscules pour « FILE », C et C++ font la distinction entre majuscule et minuscule (on dit qu’ils sont sensibles à la casse).

Pour pouvoir manipuler un fichier, un programme a besoin d'un certain nombre d'informations :

- **l'adresse** de l'endroit de la mémoire-tampon où se trouve le fichier,
- **la position** de la tête de lecture,
- **le mode d'accès** au fichier (lecture ou écriture) ...

Ouverture d’un fichier :

Fichier = fopen(nom_fichier, mode) ;

nom_fichier et mode sont de type chaine de caractère.

Avant de lire ou d'écrire dans un fichier, on notifie son accès par la commande *fopen*. Cette fonction prend comme argument le nom du fichier, négocie avec le système d'exploitation et initialise un flot de données, qui sera ensuite utilisé lors de l'écriture ou de la lecture. Après les traitements, on annule la liaison entre le fichier et le flot de données grâce à la fonction *fclose*.

Les différents modes possibles :

Mode	Description
"r"	ouverture d'un fichier texte en lecture
"w"	ouverture d'un fichier texte en écriture
"a"	ouverture d'un fichier texte en écriture à la fin
"rb"	ouverture d'un fichier binaire en lecture
"wb"	ouverture d'un fichier binaire en écriture
"ab"	ouverture d'un fichier binaire en écriture à la fin
"r+"	ouverture d'un fichier texte en lecture/écriture
"w+"	ouverture d'un fichier texte en lecture/écriture

"a+"	ouverture d'un fichier texte en lecture/écriture à la fin
"r+b"	ouverture d'un fichier binaire en lecture/écriture
"w+b"	ouverture d'un fichier binaire en lecture/écriture
"a+b"	ouverture d'un fichier binaire en lecture/écriture à la fin

Ces modes d'accès ont pour particularités :

- Si le mode contient la lettre **r**, le fichier doit exister.
- Si le mode contient la lettre **w**, le fichier peut ne pas exister. Dans ce cas, il sera créé. Si le fichier existe déjà, son ancien contenu sera perdu.
- Si le mode contient la lettre **a**, le fichier peut ne pas exister. Dans ce cas, il sera créé. Si le fichier existe déjà, les nouvelles données seront ajoutées à la fin du fichier précédent.

Trois flots standards peuvent être utilisés en C sans qu'il soit nécessaire de les ouvrir ou de les fermer :

- `stdin` (standard input) : unité d'entrée (par défaut, le clavier) ;
- `stdout` (standard output) : unité de sortie (par défaut, l'écran) ;
- `stderr` (standard error) : unité d'affichage des messages d'erreur (par défaut, l'écran).

Remarque :

Il est fortement conseillé d'afficher systématiquement les messages d'erreur sur `stderr` afin que ces messages apparaissent à l'écran même lorsque la sortie standard est redirigée.

La fonction `fclose`

Elle permet de fermer le flot qui a été associé à un fichier par la fonction `fopen`. Sa syntaxe est :

```
fclose(flot);
```

où *`flot`* est le flot de type `FILE*` retourné par la fonction `fopen` correspondant. La fonction `fclose` retourne un entier qui vaut zéro si l'opération s'est déroulée normalement (et une valeur non nulle en cas d'erreur).

Les entrées-sorties formatées

La fonction d'écriture `fprintf`

La fonction `fprintf`, analogue à `printf`, permet d'écrire des données dans un fichier. Sa syntaxe est

```
fprintf(flot,"chaîne de contrôle",expression-1, ...,
                    expression-n);
```

où *`flot`* est le flot de données retourné par la fonction `fopen`. Les spécifications de format utilisées pour la fonction `fprintf` sont les mêmes que pour `printf`.

La fonction de saisie `fscanf`

La fonction `fscanf`, analogue à `scanf`, permet de lire des données dans un fichier. Sa syntaxe est semblable à celle de `scanf` :

```
fscanf(flot,"chaîne de contrôle",argument-1,...,argument-n);
```

où *`flot`* est le flot de données retourné par `fopen`. Les spécifications de format sont ici les mêmes que celles de la fonction `scanf`.

Impression et lecture de caractères

les fonctions `fgetc` et `fputc` permettent respectivement de lire et d'écrire un caractère dans un fichier. La fonction `fgetc`, de type `int`, retourne le caractère lu dans le fichier. Elle retourne la constante `EOF` lorsqu'elle détecte la fin du fichier. Son prototype est :

```
int fgetc(FILE* flot);
```

où *`flot`* est le flot de type `FILE*` retourné par la fonction `fopen`. Comme pour la fonction `getchar`, il est conseillé de déclarer de type `int` la variable destinée à recevoir la valeur de retour de `fgetc` pour pouvoir détecter correctement la fin de fichier.

La fonction `fputc` écrit *`caractere`* dans le flot de données :

```
int fputc(int caractere, FILE *flot)
```

Elle retourne l'entier correspondant au caractère lu (ou la constante `EOF` en cas d'erreur).

Exemple :

le programme suivant lit le contenu du fichier texte `entree`, et le recopie caractère par caractère dans le fichier `sortie` :

```
#include <stdio.h>
#include <stdlib.h>
#define ENTREE "entree.txt"
#define SORTIE "sortie.txt"

int main(void)
{
  FILE *f_in, *f_out;
  int c;

  if ((f_in = fopen(ENTREE,"r")) == NULL)
    {
      fprintf(stderr, "\nErreur: Impossible de lire le fichier
      %s\n",ENTREE);
      return(EXIT_FAILURE);
    }
  if ((f_out = fopen(SORTIE,"w")) == NULL)
    {
      fprintf(stderr, "\nErreur: Impossible d'ecrire dans le fichier %s\n",
      SORTIE);
      return(EXIT_FAILURE);
    }
  while ((c = fgetc(f_in)) != EOF)
    fputc(c, f_out);
  fclose(f_in);
  fclose(f_out);
  return(EXIT_SUCCESS);
}
```

Relecture d'un caractère

Il est possible de replacer un caractère dans un flot au moyen de la fonction `ungetc` :

```
int ungetc(int caractere, FILE *flot);
```

Cette fonction place le caractère *`caractere`* (converti en `unsigned char`) dans le flot *`flot`*. En particulier, si *`caractere`* est égal au dernier caractère lu dans le flot, elle annule le déplacement provoqué par la lecture précédente. Toutefois, `ungetc` peut être utilisée avec n'importe quel caractère (sauf `EOF`). Par exemple, l'exécution du programme suivant :

```
#include <stdio.h>
#include <stdlib.h>
#define ENTREE "entree.txt"

int main(void)
{
  FILE *f_in;
  int c;

  if ((f_in = fopen(ENTREE,"r")) == NULL)
```

```
    {
      fprintf(stderr, "\nErreur: Impossible de lire le fichier
%s\n",ENTREE);
      return(EXIT_FAILURE);
    }

  while ((c = fgetc(f_in)) != EOF)
    {
      if (c == '0')
        ungetc('.',f_in);
      putchar(c);
    }
  fclose(f_in);
  return(EXIT_SUCCESS);
}
```

Sur le fichier `entree.txt` dont le contenu est `097023` affiche à l'écran `0.970.23`

Les entrées-sorties binaires

Les fonctions d'entrées-sorties binaires permettent de transférer des données dans un fichier sans transcodage. Elles sont donc plus efficaces que les fonctions d'entrée-sortie standard, mais les fichiers produits ne sont pas portables puisque le codage des données dépend des machines.

Elles sont notamment utiles pour manipuler des données de grande taille ou ayant un type composé. Leurs prototypes sont :

```
size_t fread(void *pointeur, size_t taille, size_t nombre, FILE *flot);
size_t fwrite(void *pointeur, size_t taille, size_t nombre, FILE *flot);
```

où `pointeur` est l'adresse du début des données à transférer, `taille` la taille des objets à transférer, `nombre` leur nombre. Rappelons que le type `size_t`, défini dans `stddef.h`, correspond au type du résultat de l'évaluation de `sizeof`. Il s'agit du plus grand type entier non signé.

La fonction `fread` lit les données sur le flot `flot` et la fonction `fwrite` les écrit. Elles retournent toutes deux le nombre de données transférées.

Positionnement dans un fichier

Les différentes fonctions d'entrées-sorties permettent d'accéder à un fichier en `mode séquentiel` : les données du fichier sont lues ou écrites les unes à la suite des autres. Il est également possible d'accéder à un fichier en `mode direct`, c'est-à-dire que l'on peut se positionner à n'importe quel endroit du fichier. La fonction `fseek` permet de se positionner à un endroit précis ; elle a pour prototype :

```
int fseek(FILE *flot, long deplacement, int origine);
```

La variable `deplacement` détermine la nouvelle position dans le fichier. Il s'agit d'un déplacement relatif par rapport à l'origine ; il est compté en nombre d'octets. La variable `origine` peut prendre trois valeurs :

- `SEEK_SET` (égale à 0) : début du fichier ;
- `SEEK_CUR` (égale à 1) : position courante ;
- `SEEK_END` (égale à 2) : fin du fichier.

La fonction `int rewind(FILE *flot);` permet de se positionner au début du fichier. Elle est équivalente à `fseek(flot, 0, SEEK_SET);`

La fonction `long ftell(FILE *flot);` retourne la position courante dans le fichier (en nombre d'octets depuis l'origine).